# Use of Spacecraft Command Language for Advanced Command and Control Applications

Tikiela L. Mims[1]
*Application Simulation & Support Software, Kennedy Space Center, FL, 32899*

**The purpose of this work is to evaluate the use of SCL in building and monitoring command and control applications in order to determine its fitness for space operations. Approximately 24,325 lines of PCG2 code was converted to SCL yielding a 90% reduction in the number of lines of code as many of the functions and scripts utilized in SCL could be ported and reused. Automated standalone testing, simulating the actual production environment, was performed in order to generalize and gauge the relative time it takes for SCL to update and write a given display. The use of SCL rules, functions, and scripts allowed the creation of several test cases permitting the detection of the amount of time it takes update a given set of measurements given the change in a globally existing CUI or CUI. It took the SCL system an average 926.09 ticks to update the entire display of 323 measurements.**

## Nomenclature

*CUI* = Constellation Unique Identifier
*PCG2* = Personal Computer Ground Operations Aerospace Language 2
*SCL* = Spacecraft Command Language

## I. Introduction

The advancement and transmission of technology has spun a revolution in the average American home. With the evolution of time, it has engendered an age in which the widespread possession and use of the device we call the personal computer has become the norm. Clearly, what was once available to a selected few is now utilized by many. Perhaps this helps to explain the recent rise in the production and development of computers and computer languages.

There are in fact several benefits to this increase, one in particular being that companies are able to choose from a wide array of languages the one they feel best fits their needs, and perhaps this helps to explain why domain-specific languages have seemed to persist over time, languages that give system engineers the opportunity to implement code in the creation of their own systems,. This is mainly due to the fact that many companies seek to adopt languages that they feel will not only fit there needs, but have potentials to evolve. Spacecraft Command Language (SCL) developed by Interface Control Systems, Inc. is a good example of a domain-specific language which has seemed to fit a company's needs while being given the opportunity to evolve.

For NASA, the expert system has seemed to meet the company's objective of reducing operational costs while offering a reliable medium through which the building and monitoring of command and control applications can be automated readily. To further expound on its use, SCL has been utilized in several missions and programs, including the Clementine mission, and the Far Ultraviolet Spectroscopic Explorer (FUSE) Program, all of which have contributed to the evolution and advancement of the present day SCL system; a system which creates an environment which is characterized by simplicity, rapidity, and portability.

[1] Application Simulation & Support Software Intern, NE-C1, Kennedy Space Center, and Columbus State University

1. spawn DPSPowerStatusCRT with V73S2001E1, V73S2002E1 now

2. set confirmation_msg to "OFF"

3. execute RampUp every 60 ticks

**Figure 1.0. SCL Grammar Examples.** *The first statement gives an example of SCL's "spawn" function, which in this example, executes the function DPSPowerStatusCRT with the given parameters. The second statement sets the given variable to a string literal "OFF." The third statement an example of SCL execute function. In this example, the function RampUp is executed every 60 ticks or every second.*

## II. Simplicity

In its capabilities in building and monitoring command and control applications, SCL creates an environment which is marked by simplicity. This simplicity becomes evident in its basic language and ability to eliminate the need for complex control procedures.

### A. Easy for the Nonprogrammer

The SCL grammar is very basic. Ideally, it is this basic language which helps to eliminate the need for skilled programmers. Definitively, ICS created a language that is in effect human-readable which makes it easier to understand the logic in any given SCL program. Many of its SCL's command, for instance, encompass English verbs and prepositions, including commands like "set," "execute," and "spawn." Fig 1.0 gives an example of three simple SCL commands which include the use of English verbs and prepositions.

Ideally, SCL's simplistic grammar and language creates an environment where SCL script, rules, and functions can be easily reused and understood in the creation of other applications; moreover, because of its simplistic approach, SCL quintessentially reduces the amount of resources and engineers needed to create and develop applications.

### B. Minimal Complexity

In addition to eradicating the need for skilled programmers, SCL's streamlined approach also reduces the need for creating complex control procedures. Definitively, the SCL language includes basic, fundamental programming constructs including those like If-Then-Else, repeats loops and case and assignment statements. Additionally, SCL supports only one data structure, the single-dimensional array; nevertheless, it minimizes the need to implement complex data structures like Linked Lists, Stacks, ArrayLists, and Priority Queues, common in many conventional languages like Java and C++ in the creation of control algorithms; in fact, control algorithms can be implemented through the creation of simple scripts, rules, and constraints. A simple SCL rule can be utilized to detect any deviations or failures in a system imitating the process of monitoring. In response to these triggers, the creation of simple SCL scripts can be scheduled to execute given a set of criteria. This minimization of complexity can also be seen in the porting of SCL to new environments. According to Interface Control Systems, Inc. (ICS) in order to transport the RTE to a new environment interface routines must be implemented in order to ensure connection to the new environment. This, according to ICS, requires the producing of simple interface routines that are in fact, limited in scope.[2] In retrospect, the single data structured language minimizes the need to create complex control algorithms creating a more simplistic development environment.

## III. Rapidity

Apart from creating an environment that exhibits a high level of simplicity, the SCL software system and language also yields a high level of rapidity in developing and monitoring command and control applications. At my

[2] www.interface.control.com

desktop, I was able to examine SCL's performance by detecting the amount of time it takes to write and update a given display application. This application included over 20,000 lines of embedded PCG2 code which needed to be converted to SCL in order to measure the reliability of the SCL system. Fig. 2.0 gives a snapshot of the actual display application.

### A. Code Reduction

The conversion of the existing PCG2 code to SCL required the creation of several SCL scripts, functions, and subroutines, and the building of an SCL database housing over 1,400 CUI's, or sensors. Markedly, the conversion from PCG2 to SCL saw a significant reduction in the number of lines of code. In fact, it only took SCL only 2,443 lines of code to rewrite the PCG2 application, primarily because SCL permits the use of parameters, functions, and subroutines. To make a more legitimate comparison, it takes Ground Operations Aerospace Language (GOAL), a language that also allows the implementation of functions and subroutines, approximately 6,439 lines of code to write the PCG2 application. Thus, when compared to GOAL, SCL yields a 62% reduction in the number of lines of code. Therefore, SCL creates an environment in which less time is spent in the coding phase of the software development cycle which ultimately speeds the process of development.

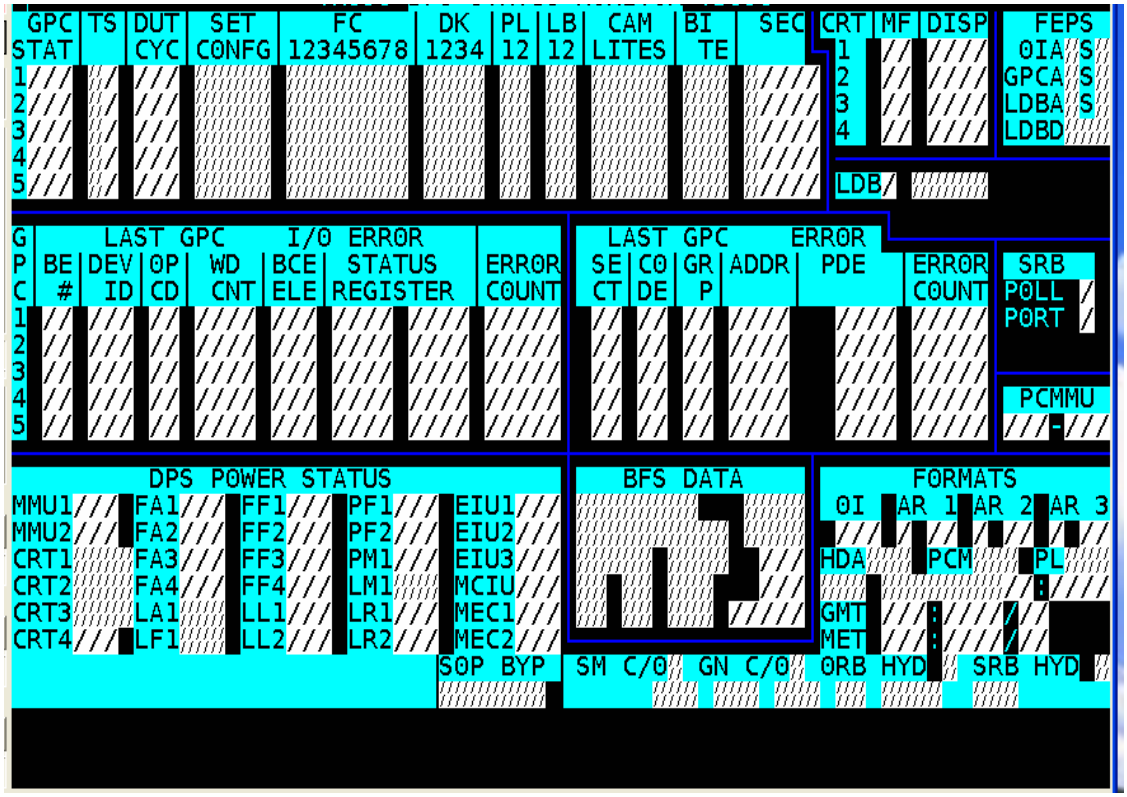


**Figure 2.0. PCG2 Display Application.** *This PCG2 display contains over 1,400 CUI's, all of which contribute in defining a particular measurement. Each column of the display contains measurements. For instance, the first column labeled "GPC STAT" contains 5 measurements. The entire display houses 323 updatable measurements.*

### B. High speed outputs

In addition to converting over 20,000 lines of PCG2 code to SCL, my job also included measuring SCL's performance in observing how fast the system performs under a particular workload. After developing a few test case scenarios in the early stages of SCL performance testing, I was able to create of a number of SCL rules and scripts which allowed me to simulate an automated testing environment. The implementation of SCL rules allowed

me to detect whether or not there were any changes in a given CUI defined in the SCL database while the execution of the command-based SCL scripts allowed me to automate directives to the SCL's Real-Time-Engine (RTE) whenever there was a change detected in a given CUI in the SCL database.

My test cases included the following: estimating the amount of time it would take SCL to update a single measurement of the display given any CUI, determining how long it would take SCL to update a given set of measurements of the display given the change in a globally existing CUI, and detecting the amount of time it would take to update the entire display. In operating on a 2.33GHz Intel Processor System with 1.96GB of RAM, the SCL system was able to conduct relatively fast updates. It took the system on the average of about 2.5 ticks or 41.6 milliseconds to update a single measurement, and SCL will yield even faster speeds when the tests are conducted in the actual production environment.

When a globally existing CUI was changed, the average amount of time that SCL took to run 2,194 lines of code to update the display was about 625.45 ticks or 10.42 seconds. Additionally in updating and rewriting the entire display application, I found that it takes SCL on the average about 926.09 ticks or 15.43 seconds to execute the 2,443 lines of SCL code. Fig. 2.1 and Fig, 2.2 illustrate examples of a compiled SCL rule and script which demonstrates the precept of command and control. For instance, in Fig. 2.1 if there is any change in the value of SGPCFIDA2 in the SCL database, the premise for the rule evaluates to *true*, and SCL's "message" command outputs the current beginning time in ticks; next, the execution of script UpdateTwo and executes its embedded three functions; afterwards, the ending time is outputted time in ticks, and the difference of the beginning time and the ending time can be utilized estimating the amount of time it takes SCL to execute the given functions.

```
rule Detect_Second_Change
subsystem none
category pwr
priority 21
activation yes
continuous yes

if change (SGPCFIDA2) then

    msg "The beginning time is " & ticks

    spawn UpdateTwo now

end if

end Detect_Second_Change
```

**Figure 2.1. SCL Rule.** *This rule detects whether or not the value of the field of the the given CUI, SGPCFIDA2, has been changed. If so, the beginning time is outputted in ticks and the script, UpdateTwo, is called and executed.*

**C. Rapidity Interpretation**

Through my conversions and tests I was able to measure the capabilities of creating and writing command and control applications. Clearly, when compared to other command and control languages like PCG2 and GOAL, SCL is the frontrunner in terms of code reduction. Performance testing allowed me to further detect the capabilities of SCL as I was able to determine whether or not the SCL software system will be able to meet performance criteria for space and ground operations. Definitively, the use of SCL rules, scripts, and constraints can be utilized to simulate command and control systems as rules can be used to monitor events, scripts to issue commands, and constraints to enforce system constraints. In retrospect, SCL's rapidity in the creation, development, and writing of applications proves that is nevertheless an ideal solution for aerospace operations as this cuts costs and time.

```
script UpdateTwo

spawn BFSDataDispMajMode with SGPCAREA2, V98U2408C1 now
spawn BFSDataDispMajMode with SGPCAREA2, V98J2299C1 now
spawn BFSDataDispMajMode with SGPCAREA2, V98J2258C1 now

msg "Ending time for Update for UpdateTwo is " & ticks

end UpdateTwo
```

**Figure 2.2. SCL Script.** *This illustration gives an example of a simple SCL Script which executes the three "spawn" directives when the rule in Figure 2.1 detects a change in the value of the database mnemonic SGPCFIDA2*

## IV. Portability

In addition to producing and providing a high level of rapidity and simplicity, the SCL software system and language also produces a high, but safe level of portability – that is, SCL components can be easily transported across several different environments while being able to adapt readily to those environments.

### A. RTE and Development Environment

SCL's Real-Time Engine (RTE) and development environment are designed specifically to be portable to a wide array of PC's and workstations. In fact, these segments are being utilized for both ground and space operations at NASA. According to Interface Control Systems, the RTE is the piece of the SCL software that is "written in C to allow ease of porting to a specific hardware platform (ground or space)."

### B. Code Reuse

In demonstrating a high level of portability the SCL system also promotes a high level of reuse. For instance, several SCL programs have been reused across the space, air, and naval industries including the SCL flight software of the Advanced System Controller Project which was reused in the building of the flight software for the 1994 Clementine program, and the Far Ultraviolet Spectroscopic Explorer (FUSE) satellite in which its SCL components were reused amongst many of its control systems. Additionally, SCL's portable and reusable features have been utilized for an assortment of aerospace applications including ground equipment software, attitude control subsystems, and satellite simulation. In rewriting the PCG2 display, I was able to reuse many of the scripts and functions in the creation of others. This essentially reduced the amount of time it took me to finish the project. Nonetheless, the reusable and portable features of SCL ultimately reduce the amount of development time to implement command and control application while reducing the costs needed for maintenance.

## V. Conclusion

There is no doubt that SCL can be utilized to build and monitor advanced command and control systems and applications. It proves to be a system that is both consistent and steadfast in its ability to automate the detection of anomalies in spacecraft data. More specifically, SCL script, subroutines, functions, constraints, and rules, give nonprogrammers the opportunity to test their own systems in an autonomous environment characterized by simplicity, rapidity, and portability. Hence, as Brian Buckley states in "SCL: An Off-The-Shelf System for Space Control", "the **flexibility** of the architecture and the open systems aspect of the SCL software gives the system broad appeal." These features ultimately suggest the reliability and feasibility of SCL in the use of advanced command and control applications; however, like any other software system, there exists areas for improvement, areas which will manifest only through careful research and further testing. One thing is for sure though, if it continues to evolve and reaches it fullest potential the SCL software system could possibly revolutionize exploration in the Constellation Program.


## Acknowledgments

The author would like to thank everyone on the NE-C1 team who assisted in the SCL learning experience.


## References


[3] Gaasbeck, Van Jim, Posner, Allan, Buckley, Brian, "Distributed Space-Segment Control Using SCL," *AAAI Press* www.interface**control**.com/marketing/pdf/flair.pdf [cited 11 December 2008].

[4] Buckley, Brian, "SCL: An Off-The-Shelf System for Space Control" NASA, Nov. 1994.